A RESOLUTION-BASED PROOF PROCEDURE
USING DELETION-DIRECTED SEARCH*

David Gelperin**

Technical Report No. 42
December '1974

* This is an updated version of
Deletion-Directed Search in Resolution-Based Proof Procedures

** Most of this work was performed at: The Ohio State University
Computer and Information Science Dept.
and was supported by National Science Foundation Grant GN-534.1
ABSTRACT

The operation of a deletion-directed search strategy for resolution-based proof procedures is discussed. The strategy attempts to determine the satisfiability of a set of input clauses while at the same time minimizing the cardinality of the set of retained clauses. Distribution, a new clause deletion rule which is fundamental to the operation of the search strategy, is also described.

Descriptive terms

Formal logic, automatic theorem proving, resolution, heuristic search, clause deletion, deletion-directed search, mate tables.

CR categories

3.60, 5.21
1. INTRODUCTION

This report describes some new techniques which can be used by resolution-based proof procedures. Since all subsequent references will be to resolution-based procedures, the adjective will be understood. In addition, familiarity with the terminology and results of J.A. Robinson\(^1\) is assumed.

In order to provide a framework for discussion, both a structural and an operational description of a proof procedure will be considered. From a structural viewpoint, a proof procedure can be denoted by a triple \(\langle \Gamma; \Delta; \Sigma \rangle\) where \(\Gamma\) is a finite, non-empty set of clause generation rules, \(\Delta\) is a finite (possibly empty) set of clause deletion rules, and \(\Sigma\) is a search strategy i.e. a procedure for applying the rules in \(\Gamma \cup \Delta\). A clause generation rule, e.g. clash resolution\(^2\), specifies the conditions constraining the clauses of an admissible resolution. A clause deletion rule, e.g. subsumption deletion, specifies the conditions under which a clause may be eliminated without affecting the unsatisfiability of a set of clauses. A search strategy, e.g. diagonal search\(^3\), sequences the generation and deletion of clauses as a proof procedure attempts to determine the satisfiability of an input clause set. Within a search strategy, clause generation is controlled by a generation strategy and clause elimination by a deletion strategy.

Turning to an operational viewpoint, a proof procedure can be characterized as a mapping which associates a finite set of clauses (i.e. the input set) with a non-empty sequence of clauses and pointers, called a trace. A trace begins with an ordered occurrence of the input set. This is followed by an ordered set of generated clauses and pointers to deleted clauses. All of the parents
of a generated clause must precede it in the trace and all deleted clauses must precede their respective pointers. Associated with each non-input trace element is a set of retained clauses which is composed of all of the clauses in the corresponding partial trace which have not been deleted i.e. those which do not have corresponding pointers in the partial trace. A trace records the operation of a proof procedure and in particular its search strategy on a specific problem.

The search space associated with a set of input clauses and a proof procedure \( \Gamma; \Delta; \Sigma \) can be represented by a labelled tree in which the input set labels the root, retained sets label the other nodes and elements of \( \Gamma \cup \Delta \) label the edges. A trace represents the particular path selected by \( \Sigma \) in the search tree. Note that the elements of \( \Gamma \cup \Delta \) are not well-defined as operators on clause sets but can be viewed as operator schemata.

Most of the work dealing with resolution has concentrated on the problem of demonstrating the unsatisfiability of an unsatisfiable set of input clauses. The problem of demonstrating the satisfiability of a satisfiable input set has been largely ignored. This is probably due, in part, to the fact that no procedure can identify all satisfiable sets. However, many solvable cases of the decision problem have been identified \(^5\) and the relationship between these cases and resolution-based proof procedures has recently been explored \(^5\). When considering applications utilizing resolution-based proof procedures, such as question-answering systems \(^6\) or robot planning systems \(^7\), situations involving the absence of sufficient information make the recognition of satisfiable sets an area of interest.
Satisfiability can be detected in three ways:

1. inspection may reveal the absence of an all positive or all negative clause,
2. a complete proof procedure may be unable to generate a new clause i.e. one which is not a variant of a clause in the retained set or
3. deletion rules may be utilized to generate the empty set of retained clauses.

In order to maximize its domain of classifiable formulas, a proof procedure must utilize deletion rules. While such rules are theoretically unnecessary in the domain of unsatisfiable formulas, in actual practice their use can help to increase the effectiveness of a proof search. Deletion rules can be used to reduce the number of candidate resolutions while at the same time preserving completeness. Careful consideration is necessary, however, since their indiscriminate use may not be cost-effective and can result in a loss of completeness.

In the next section, a new clause deletion rule is introduced and an example in Section 3 shows that this rule allows a demonstration of satisfiability not previously possible. The main topic of Section 3 is the operation of a deletion-directed search strategy which is built around the new deletion rule and attempts to minimize the cardinality of retained sets.
2. DISTRIBUTION

In order to motivate the formal presentation below, consider the unsatisfiable set \( S = \{ AB, \overline{AB}, AC, BC, \overline{C} \} \). Notice that each binary resolution which involves the literal occurrence \( A \) in \( AB \) produces a clause which is already in \( S \). This observation, viewed as a generalization of the notion of a pure literal\(^1\), suggests that \( AB \) can be deleted from \( S \) without affecting unsatisfiability. A new deletion rule validates this conjecture and the definitions which follow formalize the observation which produced it.

Associated with every ordered pair \( \langle C, D \rangle \) of clauses is a finite set (possibly empty) of ordered triples \( \langle L, M, N \rangle \) where \( L, M, \) and \( N \) are all non-empty finite sets of literals. These ordered triples are called key triples of \( \langle C, D \rangle \) and their component sets satisfy the following properties:

1. \( L \subseteq C \)
2. \( M \subseteq D \)
3. \( N = L \cap C \cup M \cap D \) is unifiable with most general unifier \( \sigma_N \) where \( L \cap C \) and \( N \cap D \) are the \( x \)-standardization and \( y \)-standardization of \( C \) and \( D \) respectively. A resolvent of \( C \) and \( D \) is any clause of the form:

\[
(C \sigma_C \cap N \cap D \sigma_D) \cup (L \sigma_L \cap M \sigma_M) \quad \text{where} \quad \langle L, M, N \rangle \quad \text{is a key triple of} \quad \langle C, D \rangle.
\]

As an example, let \( C = \text{PaFxQ}x1 \text{Pny}1 \) and \( D = \text{PaPf}x1 \) be two parent clauses. Their resolvents are \( \text{F}x1 \text{Q}x1 \text{P}y1, \text{Qag}a\text{Pf}y1, \) and \( \text{PaQf}y1 \text{P}y1 \text{Pa} \) with corresponding key triples \( \langle \{ \text{Pa} \}, \{ \text{Pa} \}, \{ \text{Fa} \} \rangle, \langle \{ \text{Pf}x \}, \{ \text{Pa} \}, \{ \text{Fa} \} \rangle \) and \( \langle \{ \text{Fa}, \text{Pf}x \}, \{ \text{Pa} \}, \{ \text{Fa}, \text{F}x1 \} \rangle \). As illustrated by the example, the definition of resolvent allows more than one key triple to correspond to a single resolvent.

The notion of a covered key triple generalizes the relationship between a key triple and its associated resolvent to include any clause which subsumes
that resolvent. Let C and D be two clauses such that \( T = \langle L, M, N \rangle \) is a key triple of \( \langle C, D \rangle \) and R is the resolvent of C and D corresponding to T. If R is subsumed by a clause in a set S of clauses, then T of \( \langle C, D \rangle \) is covered in S.

An occurrence of a literal is termed exhausted when a particular set of triples is covered. Let S be a set of clauses. Let E be a literal in a clause C. Let P be the set of all key triples T of \( \langle C, D \rangle \), where D is any clause in S-{C}, which satisfy the following conditions:

1. T has the form \( \langle \{E\}, M, N \rangle \)
2. The most general unifier \( \sigma_N \) associated with T is such that \( D_nD\sigma_N \) is not tautology.

If all key triples of P associated with non-tautologous resolvents are covered in S-{C}, then the occurrence of E in C is exhausted in S. Informally, the idea behind exhaustion is that all of the potentially useful resolvents corresponding to a particular literal occurrence are subsumed by the current set of clauses.

A clause C is distributed in a set S of clauses iff some literal in C is exhausted in S. In other words, if (almost) all of the non-tautologous resolvents that can be generated from C by only cancelling instances of E are subsumed in S-{C}, then E is exhausted and C is distributed. The 'almost' results from the fact that a non-tautologous resolvent of C and D which involves a tautologous unification instance of D need not be considered.
To clarify the above definitions, consider the set
\[ S = \{ \overline{P}xy, \overline{R}xa, Pxs, Pxs \}. \]
The key triple \( T = \langle \{Rxy\}, \{\overline{R}xa\}, \{Rx_1x_2, R_y_1\} \rangle \)
of \( \langle \overline{P}xy, \overline{R}xa \rangle \) is covered in \( S \) because \( \overline{P}xs \) subsumes the corresponding resolvent \( \overline{P}x_1\overline{Q}aSx_1 \). \( Rxy \) is exhausted in \( S \) because the only key triple of the form \( \langle \{Rxy\}, M, N \rangle \) i.e. \( T \), is covered in \( S-\{\overline{P}xy\} \). Similarly, \( \overline{R}xa \) is exhausted in \( S \). \( Qy \) is exhausted in \( S \) because all key triples \( \langle \{Qy\}, M, N \rangle \) i.e. none (it is pure), are covered in \( S \). Both literals in \( \overline{P}xs \) are exhausted in \( S \) since neither has a corresponding non-tautologous resolvent. Each of the clauses \( \overline{P}xy, \overline{R}xa, \) and \( \overline{P}xs \) is distributed in \( S \) because each contains at least one of the above mentioned exhausted literals. \( Pxs \) is not distributed in \( S \).

The concept of a distributed clause is utilized in the \textbf{Distribution Theorem} which states: If \( S \) is a set of clauses and \( C \) is a clause which is distributed in \( S \), then \( S \) is unsatisfiable iff \( S-\{C\} \) is unsatisfiable.

This theorem says that if any literal in a clause is exhausted, then the whole clause may be deleted without affecting unsatisfiability.

In the next section, a search strategy is described which is built around the notion of exhaustion. The strategy operates by performing those resolutions which cause some nearly exhausted literal occurrence to become exhausted, i.e. the strategy generates distributed clauses.
3. DELETION-DIRECTED SEARCH

The following definitions help to describe a deletion-directed proof procedure. Let a basic factor of a clause $C$ be any clause $C\theta$ where $\theta$ is a mgu of exactly two literals in $C$. $C\theta$ is a **positive basic factor** of $C$ iff the two unifying literals are positive. A clause $C$ is termed **condensed** iff no basic factor of $C$ subsumes $C$. For example, $PaxPxa$ is condensed while $PaaPxa$ is not. Let **simple binary resolution** denote a generation rule with the constraint that only one literal occurrence from each parent can be selected for a unification set, i.e. the first two components of all key triples are unit sets. Let PRO denote a proof procedure utilizing

\(<\text{positive basic factoring, condensing, simple binary resolution; subsumption deletion, distribution deletion; and deletion-directed search}>\).

Consider the following unsatisfiable set of clauses which resulted from an attempt to prove the proposition: If a diagonal of a trapezoid bisects a lower base angle, then the corresponding upper inscribed triangle is isosceles.

1. $Tabd$
2. $Tabcd$
3. $Eabddbc$
4. $TtwxyzPwzxy$
5. $PwxyzEwxyxyz$
6. $ExyzzxyIxyz$
7. $ETrstxyzEuvwxzErstuvw$

PRO would generate the following modified trace:

8. $Eabdadb$
   Delete (1,6)
9. $Padbc$
   Delete (2,4)
At this point, the retained set contains

3. $E_a b d b d c$
7. $\bar{E}_r s t v w x y z \bar{E}_r s t u v w$
10. $E_a b d b d c$
11. $\bar{E}_a b d x y z \bar{E}_a d b x y z$

Continuation of the procedure yields

12. $\bar{E}_a d b d b c$
13. $\square$

The principle which guides deletion-directed search can be stated as follows: Given a set of clauses to which no deletion rule can be applied, attempt to derive another set of clauses by adding as few resolvents as possible such that at least one of the original clauses can be deleted from the new set. The following description of the search strategy assumes a PRO context.

Given an input set, search is initiated by building a data structure called a mate table (cf. classification trees). Two literals $L_1$ and $L_2$ are mates iff there exist substitutions $\theta_1$ and $\theta_2$ such that $L_1\theta_1$ and $L_2\theta_2$ are complementary. The pointer portion of the initial mate table for the geometry example is:

\[
\begin{array}{cccccccc}
1.1 & 2.1 & 3.1 & 4.1 & 4.2 & 5.1 & 5.2 & 6.1 & 6.2 \\
6.2 & 4.1 & 7.1 & 2.1 & 5.1 & 4.2 & 6.1 & 5.2 & 1.1 \\
\end{array}
\]
A mate table records the results of a limited one-level look-a-head process which seeks a most nearly exhausted literal occurrence and in case of ties, a best one defined by tie-breaking rules. There is a one-to-one correspondence between the columns of the table and the set of safe literal occurrences. A literal occurrence \( L \) in a clause \( C \) is a safe occurrence iff \( L \) has no mates in \( C \). In our example, all literal occurrences in the first six clauses are safe while all occurrences in the seventh clause are unsafe. The following conditions are necessary but not sufficient for the creation of an entry for a safe occurrence \( L \) in a clause \( C \):

1. \( C \) resolves on \( L \) with some clause \( D \neq C \)
2. Neither the resolvent of \( C \) and \( D \) nor the associated instance of \( D \) is a tautology
3. The resolvent is not subsumed by \( D \).

The entry identifies a mate literal in \( D \) and contains the corresponding resolvent.

A table is constructed strictly by rows. The entries are sought from left to right except in the second row where the sequence of columns may skip in a search for double elimination.

In the geometry example, PRO found no pure literals while filling the first row and therefore starts the second. Finding that 1.1 has no other mates, it skips to 6.2 and checks for additional mates. If one had been found, the second row would have been continued in the hope of finding a situation like the one already found i.e. 6.2 is the only mate of 1.1 and 1.1 is the only mate of 6.2.
By adding the corresponding resolvent which is already stored in the table to the retained set, a new set is created in which both parent clauses contain an exhausted literal. Since a double elimination is the best result that can be predicted, the corresponding resolvent is added. After deletion, the mate table is updated and becomes

\[
\begin{array}{cccccccc}
2.1 & 3.1 & 4.1 & 4.2 & 5.1 & 5.2 & 8.1 \\
4.1 & 7.1 & 2.1 & 5.1 & 4.2 & 7.1 & 7.3 \\
+   & +   &     &     &     &     & \\
\end{array}
\]

Another double elimination is detected and the corresponding resolvent is added to the retained set.

\[
\begin{array}{cccccccc}
3.1 & 5.1 & 5.2 & 8.1 & 9.1 \\
7.1 & 9.1 & 7.1 & 7.3 & 5.1 \\
7.2  & +   & +   &     &     \\
\end{array}
\]

After the addition of clause 10, the table becomes

\[
\begin{array}{cccc}
3.1 & 8.1 & 10.1 \\
7.1 & 7.3 & 7.1 \\
7.2  & +   & 7.2 \\
\end{array}
\]

At this point no double eliminations are predictable, but an immediate single elimination results from adding the resolvent of clauses 7 and 8 to the retained set. During the subsequent table updating, 3 and 11 yield a unit clause. The immediate testing of all newly generated unit clauses (the end test) yields the null clause. If the end test was not used, the table would become

\[
\begin{array}{cccccccc}
3.1 & 10.1 & 11.1 & 11.2 \\
7.1 & 7.1 & 3.1 & 10.1 \\
7.2 & 7.2 & 7.3 & 7.3 \\
11.1 & 11.2 & + & + \\
\end{array}
\]
The table now indicates that the addition of two resolvents to the retained set is necessary and sufficient for a distribution deletion.

Well-known features such as clause length and functional nesting can be used to select among terminal columns e.g. 11.1 and 11.2, when deciding which resolvent to add.

In the realm of satisfiable sets, consider the following problem which results when \( \text{AT}(x, \text{under-bananas, } s_0) \) is omitted from Green's formulation\(^{10}\) of the monkey and bananas problem.

1. \( \text{HAS}(\text{monkey, bananas, } x) \)
2. \( \text{MOVABLE}(\text{box}) \)
3. \( \text{AT}(\text{box, place, } s_0) \)
4. \( \text{CLIMBABLE}(\text{monkey, box, } x) \)
5. \( \text{REACHABLE}(x, y, z) \text{HAS}(x, y, \text{reach}(x, y, z)) \)
6. \( \text{AT}(x, y, z) \text{CLIMBABLE}(w, x, z) \text{AT}(x, y, \text{climb}(w, x, z)) \)
7. \( \text{AT}(x, y, z) \text{CLIMBABLE}(w, x, z) \text{ON}(w, x, \text{climb}(w, x, z)) \)
8. \( \text{AT}(\text{box, under-bananas, } x) \text{ON}(\text{monkey, box, } x) \text{REACHABLE}(\text{monkey, bananas, } x) \)
9. \( \text{AT}(w, x, z) \text{MOVABLE}(w) \text{AT}(\text{skl}(w, x, y, z), y, z) \text{AT}(w, y, \text{move}(\text{monkey, } w, y, z)) \)
10. \( \text{AT}(w, x, z) \text{MOVABLE}(w) \text{AT}(\text{skl}(w, x, y, z), y, z) \text{AT}(\text{monkey, y, move}(\text{monkey, } w, y, z)) \)

PRO would produce the following modified trace. All of the clauses generated while processing the mate table do not appear in the trace.

11. \( \text{REACHABLE}(\text{monkey, bananas, } z) \) \( \langle 1, 5 \rangle \)
Delete \( (1, 5) \)
12. \( \text{AT}(\text{box, under-bananas, } x) \) \( \text{ON}(\text{monkey, box, } x) \) \( \langle 8, 11 \rangle \)
Delete \( (8, 11) \)
13. \( \overline{AT}(box,y,z) \)  
\[ \text{CLIMBABLE (monkey, box, z)} \]
\[ \overline{AT} (box, under-bananas, climb (monkey, box, z)) \]
Delete (7, 12)

14. \( \overline{AT}(box,y,z) \overline{AT}(box,y,\text{climb (monkey, box, z)}) \)  
Delete (6)

15. \( \overline{AT}(box,y,z) \overline{AT} (box, under-bananas, climb (monkey, box, z)) \)
Delete (4, 13)

16. \( \overline{AT}(box, under-bananas, z) \overline{AT}(box, y, z) \)  
Delete (15)

17. \( \overline{AT}(box, under-bananas, z) \) condensate of 16
Delete (16)

18. \( \overline{AT}(box,x,z) \overline{AT} (skl (box,x,y,z), y, z) \)
\[ \overline{AT} (box,y, \text{move (monkey, box, y, z)}) \]
Delete (9); (10); (2); (18); (14); (3); (17)

The satisfiability of the input set has been demonstrated since the retained set is empty. The notation following clause 18 means that the deletion of clause 9 causes a literal in clause 10 to be exhausted and therefore permits the deletion of that clause. In the same way, the deletion of clause 10 permits the deletion of clause 2, etc.

Although not illustrated by either of the examples above, deletion-directed search must be constrained by a level bound. As in the case of the unit preference strategy\(^{11}\), a level bound must be imposed in order to avoid an infinite depth-first search.

Now consider whether deletion-directed search can always find a literal occurrence to exhaust. Can resolvents always be added to a set of clauses so that some original clause will be distributed? While the answer is "no" e.g. \( \{P邢, P邢gx\} \), all of the counter-examples contain neither an all
positive nor an all negative clause and are therefore easily recognized as satisfiable sets. For all other sets, some identifiable literal occurrence is exhaustible and the required resolvents are identifiable and of finite number. Therefore, for all unsatisfiable sets, deletion-directed search will always be able to find an exhaustible literal occurrence.

The preceding examples outlined the operation of a deletion-directed strategy, but they did not reveal why and when it works efficiently. The following observations give some insight into these matters. Consider an arbitrary unsatisfiable set $S$ of clauses. If some literal occurrence $L$ in a clause $C$ has only one mate, then a resolution involving $L$ and its mate (or a descendant of that mate) must occur in every refutation of $S$ which contains $C$. If every refutation of $S$ contains $C$ (e.g. $C$ is the negation of the theorem) then a resolution on $L$ is essential to a refutation of $S$. In a minimal unsatisfiable set, any resolution which allows double parent elimination is essential and the eliminations preserve minimality. Any resolution which allows single parent elimination by distribution in a minimal unsatisfiable set is essential but the elimination may not preserve minimality. A sequence of resolutions starting from a minimal unsatisfiable set and culminating in a deletion by distribution contains at least one essential resolution.

These observations suggest that deletion-directed search will be most efficient for a minimal unsatisfiable set containing many single-mate literals -- as in the first example. Its efficiency will diminish, however, as the set in which it operates becomes increasingly non-minimal and as the population of literals with only a few mates decreases.
If necessary, the effect on non-minimality can be substantially mitigated by using a clause generation rule which is restricted by set of support constraints\textsuperscript{12}. The effect of the absence of literals having only a few mates is much more serious since the mate structure is the dominant source of guidance information. When the mate structure doesn't provide effective guidance, not only is deletion-directed search blind, but it also fails to restrict the growth of the set of retained clauses. The appropriate action in such a situation is not clear but the use of some other strategy is probably the best course.

A procedure such as PRO could be used as a front-end as well as a stand-alone system by bounding the growth of the retained set. The bounded procedure would solve some problems and reformulate the remainder. The reformulation or the original would then be passed to the next strategy.

The reader who is familiar with the Davis-Putnam procedure for testing the consistency of propositional calculus expressions\textsuperscript{13} may observe that the notion of deletion-directed search can be viewed as a generalization to the predicate calculus of their rule for eliminating atomic formulas. Other approaches to a general notion of deletion-directed search can be found in reports by B. Meltzer\textsuperscript{14}, R. Reiter\textsuperscript{15}, and R. Kowalski\textsuperscript{16}.


